Universal Hash Families

Emin Karayel

March 24, 2023

Abstract

A k-universal hash family is a probability space of functions, which have uniform distribution and form k-wise independent random variables.

They can often be used in place of classic (or cryptographic) hash functions and allow the rigorous analysis of the performance of randomized algorithms and data structures that rely on hash functions.

In 1981 Wegman and Carter [4] introduced a generic construction for such families with arbitrary k using polynomials over a finite field. This entry contains a formalization of them and establishes the property of k-universality.

To be useful the formalization also provides an explicit construction of finite fields using the factor ring of integers modulo a prime. Additionally, some generic results about independent families are shown that might be of independent interest.

1 Introduction and Definition

theory *Definitions*
 imports *HOL−Probability.Independent-Family*
begin

Universal hash families are commonly used in randomized algorithms and data structures to randomize the input of algorithms, such that probabilistic methods can be employed without requiring any assumptions about the input distribution.

If we regard a family of hash functions from a domain D to a finite range R as a uniform probability space, then the family is k-universal if:

- For each $x \in D$ the evaluation of the functions at x forms a uniformly distributed random variable on R.
- The evaluation random variables for k or fewer distinct domain elements form an independent family of random variables.

This definition closely follows the definition from Vadhan [3, §3.5.5], with the minor modification that independence is required not only for exactly k, but also for *fewer* than k distinct domain elements. The correction is due to the fact that in the corner case where D has fewer than k elements, the second part of their definition becomes void. In the formalization this helps avoid an unnecessary assumption in the theorems.

The following definition introduces the notion of k-wise independent random variables:

definition (**in** *prob-space*) *k-wise-indep-vars* **where**
  k-wise-indep-vars k M′ X I =
    ($\forall$ *J* $\subseteq$ *I*. *card J* $\leq$ *k* $\longrightarrow$ *finite J* $\longrightarrow$ *indep-vars M′ X J*)

lemma (**in** *prob-space*) *k-wise-indep-vars-subset*:
  assumes *k-wise-indep-vars k M′ X I*
  assumes *J* $\subseteq$ *I*
  assumes *finite J*
  assumes *card J* $\leq$ *k*
  shows *indep-vars M′ X J*
  using *assms*
  by (*simp add*:*k-wise-indep-vars-def*)

Similarly for a finite non-empty set A the predicate *uniform-on X A* indicates that the random variable is uniformly distributed on A:

definition (**in** *prob-space*) *uniform-on X A* = (
  distr M (*count-space UNIV*) *X* = *uniform-measure* (*count-space UNIV*) *A* $\wedge$
  A $\neq$ {} $\wedge$ *finite A* $\wedge$ *random-variable* (*count-space UNIV*) *X*)

lemma (**in** *prob-space*) *uniform-onD*:
  assumes *uniform-on X A*
  shows *prob* {ω $\in$ *space M*. *X* ω $\in$ *B*} = *card* (*A* $\cap$ *B*) / *card A*
proof −
  have *prob* {ω $\in$ *space M*. *X* ω $\in$ *B*} = *prob* (*X* −‘ *B* $\cap$ *space M*)
    by (*subst Int-commute*, *simp add*:*vimage-def Int-def*)
  also have ... = *measure* (*distr M* (*count-space UNIV*) *X*) *B*
    using *assms* **by** (*subst measure-distr*, *auto simp*:*uniform-on-def*)
  also have ... = *measure* (*uniform-measure* (*count-space UNIV*) *A*) *B*
    using *assms* **by** (*simp add*:*uniform-on-def*)
  also have ... = *card* (*A* $\cap$ *B*) / *card A*
    using *assms* **by** (*subst measure-uniform-measure*, *auto simp*:*uniform-on-def*)+
  finally show *?thesis* **by** *simp*
qed

With the two previous definitions it is possible to define the k-universality condition for a family of hash functions from D to R:

definition (**in** *prob-space*) *k-universal k X D R* = (
  k-wise-indep-vars k (λ-. *count-space UNIV*) *X D* $\wedge$
  ($\forall$ *i* $\in$ *D*. *uniform-on* (*X i*) *R*))

Note: The definition is slightly more generic then the informal specification from above. This is because usually a family is formed by a single function with a variable seed parameter. Instead of choosing a random function from a probability space, a random seed is chosen from the probability space which parameterizes the hash function.

The following section contains some preliminary results about independent families of random variables. Section 3 introduces the Carter-Wegman hash family, which is an explicit construction of k-universal families for arbitrary k using polynomials over finite fields. The last section contains a proof that the factor ring of the integers modulo a prime ideal is a finite field, followed by an isomorphic construction of prime fields over an initial segment of the natural numbers.

end

2 Preliminary Results

theory *Preliminary-Results*
  imports
    Definitions
    HOL−Probability.Stream-Space
    HOL−Probability.Probability-Mass-Function
begin

lemma *set-comp-image-cong*:
  assumes $\bigwedge x.\ P\ x \Longrightarrow f\ x = h\ (g\ x)$
  shows $\{f\ x|\ x.\ P\ x\} = h\ `\ \{g\ x|\ x.\ P\ x\}$
  using *assms* **by** (*auto simp*: *setcompr-eq-image*)

lemma (**in** *prob-space*) *k-wise-indep-vars-compose*:
  assumes *k-wise-indep-vars* $k\ M'\ X\ I$
  assumes $\bigwedge i.\ i \in I \Longrightarrow Y\ i \in$ *measurable* $(M'\ i)\ (N\ i)$
  shows *k-wise-indep-vars* $k\ N\ (\lambda i\ x.\ Y\ i\ (X\ i\ x))\ I$
  using *indep-vars-compose2*[**where** $N{=}N$ **and** $X{=}X$ **and** $Y{=}Y$ **and** $M'{=}M'$] *assms*
  by (*simp add*: *k-wise-indep-vars-def subsetD*)

The following two lemmas are of independent interest, they help infer independence of events and random variables on distributions. (Candidates for *HOL−Probability.Independent-Family*).

lemma (**in** *prob-space*) *indep-sets-distr*:
  fixes A
  assumes *random-variable* $N\ f$
  defines $F \equiv (\lambda i.\ (\lambda a.\ f\ -`\ a \cap$ *space* $M)\ `\ A\ i)$
  assumes *indep-F*: *indep-sets* $F\ I$
  assumes *sets-A*: $\bigwedge i.\ i \in I \Longrightarrow A\ i \subseteq$ *sets* N
  shows *prob-space.indep-sets* (*distr* $M\ N\ f$) $A\ I$

proof (*rule prob-space.indep-setsI*)
  show $\bigwedge A'\ J.\ J \neq \{\} \Longrightarrow J \subseteq I \Longrightarrow \mathit{finite}\ J \Longrightarrow \forall j{\in}J.\ A'\ j \in A\ j \Longrightarrow$
    $\mathit{measure}\ (\mathit{distr}\ M\ N\ f)\ (\bigcap\ (A'\ `\ J)) = (\prod j{\in}J.\ \mathit{measure}\ (\mathit{distr}\ M\ N\ f)\ (A'\ j))$
  proof −
    fix $A'\ J$
    assume $a{:}J \subseteq I\ \mathit{finite}\ J\ J \neq \{\}\ \forall j \in J.\ A'\ j \in A\ j$

    define F' **where** $F' = (\lambda i.\ f\ -`\ A'\ i \cap \mathit{space}\ M)$

    have $\bigcap\ (F'\ `\ J) = f\ -`\ (\bigcap\ (A'\ `\ J)) \cap \mathit{space}\ M$
      unfolding *set-eq-iff* F'*-def* **using** $a(3)$ **by** *simp*
    moreover have $\bigcap\ (A'\ `\ J) \in \mathit{sets}\ N$
      by (*metis a sets-A sets.finite-INT subset-iff*)
    ultimately have b:
      $\mathit{measure}\ (\mathit{distr}\ M\ N\ f)\ (\bigcap\ (A'\ `\ J)) = \mathit{measure}\ M\ (\bigcap\ (F'\ `\ J))$
      by (*metis assms*(*1*) *measure-distr*)

    have $\bigwedge j.\ j \in J \Longrightarrow F'\ j \in F\ j$
      using $a(4)$ F'*-def F-def* **by** *blast*
    hence $c{:}\mathit{measure}\ M\ (\bigcap\ (F'\ `\ J)) = (\prod j{\in}\ J.\ \mathit{measure}\ M\ (F'\ j))$
      by (*metis indep-F indep-setsD a*(*1*,*2*,*3*))

    have $\bigwedge j.\ j \in J \Longrightarrow F'\ j = \ f\ -`\ A'\ j\ \cap \mathit{space}\ M$
      by (*simp add*:F'*-def*)
    moreover have $\bigwedge j.\ j \in J \Longrightarrow A'\ j \in \mathit{sets}\ N$
      using $a(1,4)$ *sets-A* **by** *blast*
    ultimately have d:
      $\bigwedge j.\ j \in J \Longrightarrow \mathit{measure}\ M\ (F'\ j) = \mathit{measure}\ (\mathit{distr}\ M\ N\ f)\ (A'\ j)$
      using *assms*(*1*) *measure-distr* **by** *metis*

    show
      $\mathit{measure}\ (\mathit{distr}\ M\ N\ f)\ (\bigcap\ (A'\ `\ J)) = (\prod j{\in}J.\ \mathit{measure}\ (\mathit{distr}\ M\ N\ f)\ (A'\ j))$
      using *b c d* **by** *auto*
  qed
  show *prob-space* (*distr M N f*) **using** *prob-space-distr assms* **by** *blast*
  show $\bigwedge i.\ i \in I \Longrightarrow A\ i \subseteq \mathit{sets}\ (\mathit{distr}\ M\ N\ f)$ **using** *sets-A sets-distr* **by** *blast*
qed

lemma (**in** *prob-space*) *indep-vars-distr*:
  assumes $f \in \mathit{measurable}\ M\ N$
  assumes $\bigwedge i.\ i \in I \Longrightarrow X'\ i \in \mathit{measurable}\ N\ (M'\ i)$
  assumes $\mathit{indep\text{-}vars}\ M'\ (\lambda i.\ (X'\ i) \circ f)\ I$
  shows $\mathit{prob\text{-}space.indep\text{-}vars}\ (\mathit{distr}\ M\ N\ f)\ M'\ X'\ I$
proof −
  interpret D: *prob-space* (*distr M N f*)
    using *prob-space-distr*[*OF assms*(*1*)] **by** *simp*

  have a: $f \in \mathit{space}\ M \to \mathit{space}\ N$ **using** *assms*(*1*) **by** (*simp add*:*measurable-def*)

have *D.indep-sets* $(\lambda i.\ \{X'\ i\ -`\ A \cap space\ N\ |A.\ A \in sets\ (M'\ i)\})\ I$
proof (*rule indep-sets-distr*[*OF assms*(*1*)])
have $\bigwedge i.\ i \in I \Longrightarrow \{(X'\ i \circ f)\ -`\ A \cap space\ M\ |A.\ A \in sets\ (M'\ i)\} =$
$(\lambda a.\ f\ -`\ a \cap space\ M)\ `\ \{X'\ i\ -`\ A \cap space\ N\ |A.\ A \in sets\ (M'\ i)\}$
by (*rule set-comp-image-cong*, *simp add*:*set-eq-iff*, *use a* **in** *blast*)
thus *indep-sets* $(\lambda i.\ (\lambda a.\ f\ -`\ a \cap space\ M)\ `$
$\{X'\ i\ -`\ A \cap space\ N\ |A.\ A \in sets\ (M'\ i)\})\ I$
using *assms*(*3*) **by** (*simp add*:*indep-vars-def2 cong*:*indep-sets-cong*)
next
fix *i*
assume $i \in I$
thus $\{X'\ i\ -`\ A \cap space\ N\ |A.\ A \in sets\ (M'\ i)\} \subseteq sets\ N$
using *assms*(*2*) *measurable-sets* **by** *blast*
qed
thus *?thesis*
using *assms* **by** (*simp add*:*D.indep-vars-def2*)
qed

lemma *range-inter*: *range* $((\cap)\ F) = Pow\ F$
unfolding *image-def* **by** *auto*

The singletons and the empty set form an intersection stable generator of a countable discrete σ-algebra:

lemma *sigma-sets-singletons-and-empty*:
assumes *countable M*
shows *sigma-sets M* $(insert\ \{\}\ ((\lambda k.\ \{k\})\ `\ M)) = Pow\ M$
proof −
have *sigma-sets M* $((\lambda k.\ \{k\})\ `\ M) = Pow\ M$
using *assms sigma-sets-singletons* **by** *auto*
hence $Pow\ M \subseteq sigma\text{-}sets\ M\ (insert\ \{\}\ ((\lambda k.\ \{k\})\ `\ M))$
by (*metis sigma-sets-subseteq subset-insertI*)
moreover have $(insert\ \{\}\ ((\lambda k.\ \{k\})\ `\ M)) \subseteq Pow\ M$ **by** *blast*
hence *sigma-sets M* $(insert\ \{\}\ ((\lambda k.\ \{k\})\ `\ M)) \subseteq Pow\ M$
by (*meson sigma-algebra.sigma-sets-subset sigma-algebra-Pow*)
ultimately show *?thesis* **by** *force*
qed

In some of the following theorems, the premise $M = measure\text{-}pmf\ p$ is used. This allows stating theorems that hold for pmfs more concisely, for example, instead of $measure\text{-}pmf.prob\ p\ A \le measure\text{-}pmf.prob\ p\ B$ we can just write $M = measure\text{-}pmf\ p \Longrightarrow prob\ A \le prob\ B$ in the locale *prob-space*.

lemma *prob-space-restrict-space*:
assumes [*simp*]:$M = measure\text{-}pmf\ p$
shows *prob-space* (*restrict-space M* (*set-pmf p*))
by (*rule prob-spaceI*, *auto simp*:*emeasure-restrict-space emeasure-pmf*)

The abbreviation below is used to specify the discrete σ-algebra on *UNIV* as a measure space. It is used in places where the existing definitions, such as *indep-vars*, expect a measure space even though only a *measurable* space is really needed, i.e., in cases where the property is invariant with respect to the actual measure.

hide-const (**open**) *discrete*

abbreviation *discrete* $\equiv$ *count-space UNIV*

lemma (**in** *prob-space*) *indep-vars-restrict-space*:
  assumes [*simp*]:M = *measure-pmf p*
  assumes
    prob-space.indep-vars (*restrict-space M* (*set-pmf p*)) (λ-. *discrete*) *X I*
  shows *indep-vars* (λ-. *discrete*) *X I*
proof –
  have *a*: *id* $\in$ *restrict-space M* (*set-pmf p*) $\to_M$ *M*
    by (*simp add*:*measurable-def range-inter sets-restrict-space*)

  have *prob-space.indep-vars* (*distr* (*restrict-space M* (*set-pmf p*)) *M id*) (λ-. *discrete*) *X I*
   using *assms a prob-space-restrict-space* **by** (*auto intro*!:*prob-space.indep-vars-distr*)
  moreover have
    $\bigwedge$*A*. *emeasure* (*distr* (*restrict-space M* (*set-pmf p*)) *M id*) *A* = *emeasure M A*
    using *emeasure-distr*[*OF a*]
    by (*auto simp add*: *emeasure-restrict-space emeasure-Int-set-pmf*)
  hence *distr* (*restrict-space M p*) *M id* = *M*
    by (*auto intro*: *measure-eqI*)
  ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *prob-space*) *measure-pmf-eq*:
  assumes *M* = *measure-pmf p*
  assumes $\bigwedge$*x*. *x* $\in$ *set-pmf p* $\Longrightarrow$ (*x* $\in$ *P*) = (*x* $\in$ *Q*)
  shows *prob P* = *prob Q*
  unfolding *assms*(*1*)
  by (*rule measure-eq-AE*, *rule AE-pmfI*[*OF assms*(*2*)], *auto*)

The following lemma is an intro rule for the independence of random variables defined on pmfs. In that case it is possible, to check the independence of random variables point-wise.

The proof relies on the fact that the support of a pmf is countable and the σ-algebra of such a set can be generated by singletons.

lemma (**in** *prob-space*) *indep-vars-pmf*:
  assumes [*simp*]:*M* = *measure-pmf p*
  assumes $\bigwedge$*a J*. *J* $\subseteq$ *I* $\Longrightarrow$ *finite J* $\Longrightarrow$
    prob $\{\omega.\ \forall i \in J.\ X\ i\ \omega = a\ i\}$ = $(\prod i \in J.\ prob\ \{\omega.\ X\ i\ \omega = a\ i\})$
  shows *indep-vars* (λ-. *discrete*) *X I*
proof –
  interpret *R*:*prob-space* (*restrict-space M* (*set-pmf p*))
    using *prob-space-restrict-space* **by** *auto*

have *events-eq-pow*: *R.events* = *Pow* (*set-pmf p*)
  by (*simp add*:*sets-restrict-space range-inter*)

define *G* **where** $G = (\lambda i.\ \{\{\}\} \cup (\lambda x.\ \{x\})$ ‘ $(X\ i$ ‘ *set-pmf p*))
define *F* **where** $F = (\lambda i.\ \{X\ i$ −‘ $a \cap$ *set-pmf p*$|a.\ a \in G\ i\})$

have *sigma-sets-pow*:
  $\bigwedge i.\ i \in I \Longrightarrow$ *sigma-sets* ($X\ i$ ‘ *set-pmf p*) ($G\ i$) = *Pow* ($X\ i$ ‘ *set-pmf p*)
 by (*simp add*:*G-def*, *metis countable-image countable-set-pmf sigma-sets-singletons-and-empty*)

have *F-in-events*: $\bigwedge i.\ i \in I \Longrightarrow F\ i \subseteq$ *Pow* (*set-pmf p*)
  unfolding *F-def* **by** *blast*

have *as-sigma-sets*:
  $\bigwedge i.\ i \in I \Longrightarrow \{u.\ \exists A.\ u = X\ i$ −‘ $A \cap$ *set-pmf p*$\}$ = *sigma-sets* (*set-pmf p*) (*F*
i)
 proof −
  fix *i*
  assume $a{:}i \in I$
  have $\bigwedge A.\ X\ i$ −‘ $A \cap$ *set-pmf p* $= X\ i$ −‘ $(A \cap X\ i$ ‘ *set-pmf p*) $\cap$ *set-pmf p*
   by *auto*
  hence $\{u.\ \exists A.\ u = X\ i$ −‘ $A \cap$ *set-pmf p*$\}$ =
     $\{X\ i$ −‘ $A \cap$ *set-pmf p* $|A.\ A \subseteq X\ i$ ‘ *set-pmf p*$\}$
   by (*metis* (*no-types*, *opaque-lifting*) *inf-le2*)
  also have
   ... = $\{X\ i$ −‘ $A \cap$ *set-pmf p* $|A.\ A \in$ *sigma-sets* ($X\ i$ ‘ *set-pmf p*) ($G\ i$)$\}$
   using *a* **by** (*simp add*:*sigma-sets-pow*)
  also have ... = *sigma-sets* (*set-pmf p*) $\{X\ i$ −‘ $a \cap$ *set-pmf p* $|a.\ a \in G\ i\}$
   by (*subst sigma-sets-vimage-commute*[*symmetric*], *auto*)
  also have ... = *sigma-sets* (*set-pmf p*) ($F\ i$)
   by (*simp add*:*F-def*)
  finally show
   $\{u.\ \exists A.\ u = X\ i$ −‘ $A \cap$ *set-pmf p*$\}$ = *sigma-sets* (*set-pmf p*) ($F\ i$)
   by *simp*
 qed

have *F-Int-stable*: $\bigwedge i.\ i \in I \Longrightarrow$ *Int-stable* ($F\ i$)
proof (*rule Int-stableI*)
  fix *i a b*
  assume $i \in I$ $a \in F\ i$ $b \in F\ i$
  thus $a \cap b \in (F\ i)$
   unfolding *F-def G-def* **by** (*cases* $a \cap b = \{\}$, *auto*)
qed

have *F-indep-sets*:*R.indep-sets F I*
proof (*rule R.indep-setsI*)
  fix *i*
  assume $i \in I$
  show $F\ i \subseteq$ *R.events*

unfolding *F-def events-eq-pow* **by** *blast*
next
fix A
fix J
assume a:$J \subseteq I$ $J \neq \{\}$ *finite* J $\forall j{\in}J.\ A\ j \in F\ j$
have b: $\bigwedge j.\ j \in J \Longrightarrow A\ j \subseteq$ *set-pmf* p
by (*metis PowD a*(*1*,*4*) *subsetD F-in-events*)
obtain x **where** *x-def*:$\bigwedge j.\ j \in J \Longrightarrow A\ j = X\ j\ -`\ x\ j \cap$ *set-pmf* $p \wedge x\ j \in G\ j$
using a **by** (*simp add*:*Pi-def F-def*, *metis*)

show *R.prob* $(\bigcap\ (A\ `\ J)) = (\prod j{\in}J.\ R.prob\ (A\ j))$
proof (*cases* $\exists j \in J.\ A\ j = \{\}$)
case *True*
hence $\bigcap\ (A\ `\ J) = \{\}$ **by** *blast*
then show *?thesis*
using *a True* **by** (*simp*, *metis measure-empty*)
next
case *False*
then have $\bigwedge j.\ j \in J \Longrightarrow x\ j \neq \{\}$ **using** *x-def* **by** *auto*
hence $\bigwedge j.\ j \in J \Longrightarrow x\ j \in (\lambda x.\ \{x\})\ `\ X\ j\ `$ *set-pmf* p
using *x-def* **by** (*simp add*:*G-def*)
then obtain y **where** *y-def*: $\bigwedge j.\ j \in J \Longrightarrow x\ j = \{y\ j\}$
by (*simp add*:*image-def*, *metis*)

have $\bigcap\ (A\ `\ J) \subseteq$ *set-pmf* p **using** b $a(2)$ **by** *blast*
hence *R.prob* $(\bigcap\ (A\ `\ J)) =$ *prob* $(\bigcap\ j \in J.\ A\ j)$
by (*simp add*: *measure-restrict-space*)
also have ... = *prob* $(\{\omega.\ \forall j \in J.\ X\ j\ \omega = y\ j\})$
using *a x-def y-def* **apply** (*simp add*:*vimage-def measure-Int-set-pmf*)
by (*rule arg-cong2* [**where** *f*=*measure*], *auto*)
also have ... = $(\prod\ j{\in}\ J.\ prob\ (A\ j))$
using *x-def y-def a assms*(*2*)
by (*simp add*:*vimage-def measure-Int-set-pmf*)
also have ... = $(\prod j{\in}J.\ R.prob\ (A\ j))$
using b **by** (*simp add*: *measure-restrict-space cong*:*prod.cong*)
finally show *?thesis* **by** *blast*
qed
qed

have *R.indep-sets* ($\lambda i.$ *sigma-sets* (*set-pmf* p) (F i)) I
using *R.indep-sets-sigma*[*simplified*] *F-Int-stable F-indep-sets*
by (*auto simp*:*space-restrict-space*)

hence *R.indep-sets* $(\lambda i.\ \{u.\ \exists A.\ u = X\ i\ -`\ A \cap$ *set-pmf* $p\})$ I
by (*simp add*: *as-sigma-sets cong*:*R.indep-sets-cong*)

hence *R.indep-vars* (λ-. *discrete*) X I
unfolding *R.indep-vars-def2*
by (*simp add*:*measurable-def sets-restrict-space range-inter*)

thus *?thesis*
using *indep-vars-restrict-space*[*OF assms*(*1*)] **by** *simp*
qed

lemma (**in** *prob-space*) *split-indep-events*:
assumes $M = \textit{measure-pmf } p$
assumes *indep-vars* (λi. *discrete*) X' I
assumes $K \subseteq I$ *finite* K
shows *prob* $\{\omega.\ \forall x \in K.\ P\ x\ (X'\ x\ \omega)\} = (\prod x \in K.\ \textit{prob}\ \{\omega.\ P\ x\ (X'\ x\ \omega)\})$
proof −
have [*simp*]: *space* M = *UNIV* *events* = *UNIV* *prob UNIV* = *1*
by (*simp add*:*assms*(*1*))+

have *indep-vars* (λ-. *discrete*) X' K
using *assms*(*2*,*3*) *indep-vars-subset* **by** *blast*
hence *indep-events* (λx. $\{\omega \in \textit{space } M.\ P\ x\ (X'\ x\ \omega)\}$) K
using *indep-eventsI-indep-vars* **by** *force*
hence *a*:*indep-events* (λx. $\{\omega.\ P\ x\ (X'\ x\ \omega)\}$) K
by *simp*

have *prob* $\{\omega.\ \forall x \in K.\ P\ x\ (X'\ x\ \omega)\}$ = *prob* $(\bigcap x \in K.\ \{\omega.\ P\ x\ (X'\ x\ \omega)\})$
by (*simp add*: *measure-pmf-eq*[*OF assms*(*1*)])
also have ... = $(\prod x \in K.\ \textit{prob}\ \{\omega.\ P\ x\ (X'\ x\ \omega)\})$
using *a assms*(*4*) **by** (*cases* $K = \{\}$, *auto simp*: *indep-events-def*)
finally show *?thesis* **by** *simp*
qed

lemma *pmf-of-set-eq-uniform*:
assumes *finite* A $A \neq \{\}$
shows *measure-pmf* (*pmf-of-set* A) = *uniform-measure discrete* A
proof −
have *a*:*real* (*card* A) > *0* **using** *assms*
by (*simp add*: *card-gt-0-iff*)

have *b*:
$\bigwedge Y$. *emeasure* (*pmf-of-set* A) Y = *emeasure* (*uniform-measure discrete* A) Y
using *assms a*
by (*simp add*: *emeasure-pmf-of-set divide-ennreal ennreal-of-nat-eq-real-of-nat*)

show *?thesis*
by (*rule measure-eqI*, *auto simp add*: *b*)
qed

lemma (**in** *prob-space*) *uniform-onI*:
assumes $M = \textit{measure-pmf } p$
assumes *finite* A $A \neq \{\}$
assumes $\bigwedge a$. *prob* $\{\omega.\ X\ \omega = a\}$ = *indicator* A a / *card* A
shows *uniform-on* X A

proof −
  have *a*:$\bigwedge$*a*. *measure-pmf.prob p* {*x*. *X x* = *a*} = *indicator A a* / *card A*
    using *assms*(*1*,*4*) **by** *simp*

  have *b*:*map-pmf X p* = *pmf-of-set A*
    by (*rule pmf-eqI*, *simp add*:*assms pmf-map vimage-def a*)

  have *distr M discrete X* = *map-pmf X p*
    by (*simp add*: *map-pmf-rep-eq assms*(*1*))
  also have ... = *measure-pmf* (*pmf-of-set A*)
    using *b* **by** *simp*
  also have ... = *uniform-measure discrete A*
    by (*rule pmf-of-set-eq-uniform*[*OF assms*(*2*,*3*)])
  finally have *distr M discrete X* = *uniform-measure discrete A*
    by *simp*
  moreover have *random-variable discrete X*
    by (*simp add*: *assms*(*1*))
  ultimately show *?thesis* **using** *assms*(*2*,*3*)
    by (*simp add*: *uniform-on-def*)
qed

end

3 Carter-Wegman Hash Family

theory *Carter-Wegman-Hash-Family*
  imports
    Interpolation-Polynomials-HOL-Algebra.Interpolation-Polynomial-Cardinalities
    Preliminary-Results
begin

The Carter-Wegman hash family is a generic method to obtain k-universal hash families for arbitrary k. (There are faster solutions, such as tabulation hashing, which are limited to a specific k. See for example [2].)

The construction was described by Wegman and Carter [4], it is a hash family between the elements of a finite field and works by choosing randomly a polynomial over the field with degree less than k. The hash function is the evaluation of a such a polynomial.

Using the property that the fraction of polynomials interpolating a given set of $s \leq k$ points is *1* / *real* (*card* (*carrier R*))^s, which is shown in [1], it is possible to obtain both that the hash functions are k-wise independent and uniformly distributed.

In the following two locales are introduced, the main reason for both is to make the statements of the theorems and proofs more concise. The first locale *poly-hash-family* fixes a finite ring R and the probability space of the polynomials of degree less than k. Because the ring is not a field, the family is not yet k-universal, but it is still possible to state a few results such as the fact that the range of the hash function is a subset of the carrier of the ring.

The second locale *carter-wegman-hash-family* is an extension of the former with the assumption that R is a field with which the k-universality follows. The reason for using two separate locales is to support use cases, where the ring is only probably a field. For example if it is the set of integers modulo an approximate prime, in such a situation a subset of the properties of an algorithm using approximate primes would need to be verified even if R is only a ring.

definition (**in** *ring*) *hash x ω = eval ω x*

locale *poly-hash-family = ring +*
  fixes *k :: nat*
  assumes *finite-carrier*[*simp*]: *finite* (*carrier R*)
  assumes *k-ge-0*: *k > 0*
begin

definition *space* **where** *space = bounded-degree-polynomials R k*
definition *M* **where** *M = measure-pmf* (*pmf-of-set space*)

lemma *finite-space*[*simp*]:*finite space*
    unfolding *space-def* **using** *fin-degree-bounded finite-carrier* **by** *simp*

lemma *non-empty-bounded-degree-polynomials*[*simp*]:*space ≠ {}*
    unfolding *space-def* **using** *non-empty-bounded-degree-polynomials* **by** *simp*

This is to add *carrier-not-empty* to the simp set in the context of *poly-hash-family*:

lemma *non-empty-carrier*[*simp*]: *carrier R ≠ {}*
  by (*simp add*:*carrier-not-empty*)

sublocale *prob-space M*
  by (*simp add*:*M-def prob-space-measure-pmf*)

lemma *hash-range*[*simp*]:
  assumes *ω ∈ space*
  assumes *x ∈ carrier R*
  shows *hash x ω ∈ carrier R*
  using *assms* **unfolding** *hash-def space-def bounded-degree-polynomials-def*
  by (*simp*, *metis eval-in-carrier polynomial-incl univ-poly-carrier*)

lemma *hash-range-2*:
  assumes *ω ∈ space*
  shows (*λx. hash x ω*) *‘ carrier R ⊆ carrier R*
  using *hash-range assms* **by** *auto*

lemma *integrable-M*[*simp*]:

fixes f :: $'a$ *list* $\Rightarrow$ $'c$::{*banach*, *second-countable-topology*}
shows *integrable M f*
unfolding *M-def*
by (*rule integrable-measure-pmf-finite*, *simp*)

end

locale *carter-wegman-hash-family* = *poly-hash-family* +
assumes *field-R*: *field R*
begin
sublocale *field*
using *field-R* **by** *simp*

abbreviation *field-size* $\equiv$ *card* (*carrier R*)

lemma *poly-cards*:
assumes $K \subseteq$ *carrier R*
assumes *card* $K \leq k$
assumes y ‘ $K \subseteq$ (*carrier R*)
shows
card $\{\omega \in$ *space*. $(\forall k \in K.$ *eval* ω $k = y$ $k)\}$ = *field-size*^($k-$*card* K)
unfolding *space-def*
using *interpolating-polynomials-card*[**where** n=$k-$*card* K **and** K=K] *assms*
using *finite-carrier finite-subset* **by** *fastforce*

lemma *poly-cards-single*:
assumes $x \in$ *carrier R*
assumes $y \in$ *carrier R*
shows *card* $\{\omega \in$ *space*. *eval* ω $x = y\}$ = *field-size*^($k-1$)
using *poly-cards*[**where** K={x} **and** y=λ-. y, *simplified*] *assms k-ge-0* **by** *simp*

lemma *hash-prob*:
assumes $K \subseteq$ *carrier R*
assumes *card* $K \leq k$
assumes y ‘ $K \subseteq$ *carrier R*
shows
prob $\{\omega.$ $(\forall x \in K.$ *hash* x $\omega = y$ $x)\}$ = $1/$(*real field-size*)^*card* K
proof −
have **0** $\in$ *carrier R* **by** *simp*

hence *a*:*field-size* > 0
using *finite-carrier card-gt-0-iff* **by** *blast*

have *b*:*real* (*card* $\{\omega \in$ *space*. $\forall x \in K.$ *eval* ω $x = y$ $x\}$) / *real* (*card space*) =
1 / *real field-size* ^ *card* K
using *a assms*(2)
apply (*simp add*: *frac-eq-eq poly-cards*[*OF assms*(1,2,3)] *power-add*[*symmetric*])
by (*simp add*:*space-def bounded-degree-polynomials-card*)

```
  show ?thesis
    unfolding M-def
    by (simp add:hash-def measure-pmf-of-set Int-def b)
qed

lemma prob-single:
  assumes x ∈ carrier R y ∈ carrier R
  shows prob {ω. hash x ω = y} = 1/(real field-size)
  using hash-prob[where K={x}] assms finite-carrier k-ge-0 by simp

lemma prob-range:
  assumes [simp]:x ∈ carrier R
  shows prob {ω. hash x ω ∈ A} = card (A ∩ carrier R) / field-size
proof –
  have prob {ω. hash x ω ∈ A} = prob (⋃ a ∈ A ∩ carrier R. {ω. hash x ω = a})
    by (rule measure-pmf-eq, auto simp:M-def)
  also have ... = (∑ a ∈ (A ∩ carrier R). prob {ω. hash x ω = a})
    by (rule measure-finite-Union, auto simp:M-def disjoint-family-on-def)
  also have ... = (∑ a ∈ (A ∩ carrier R). 1/(real field-size))
    by (rule sum.cong, auto simp:prob-single)
  also have ... = card (A ∩ carrier R) / field-size
    by simp
  finally show ?thesis by simp
qed

lemma indep:
  assumes J ⊆ carrier R
  assumes card J ≤ k
  shows indep-vars (λ-. discrete) hash J
proof –
  have 0 ∈ carrier R by simp
  hence card-R-ge-0:field-size > 0
    using card-gt-0-iff finite-carrier by blast

  have fin-J: finite J
    using finite-carrier assms(1) finite-subset by blast

  show ?thesis
  proof (rule indep-vars-pmf[OF M-def])
    fix a
    fix J'
    assume a: J' ⊆ J finite J'
    have card-J': card J' ≤ k
      by (metis card-mono order-trans a(1) assms(2) fin-J)
    have J'-in-carr: J' ⊆ carrier R by (metis order-trans a(1) assms(1))

    show prob {ω. ∀x∈J'. hash x ω = a x} = (∏x∈J'. prob {ω. hash x ω = a x})
    proof (cases a ` J' ⊆ carrier R)
      case True
```

```
      have a-carr: ⋀x. x ∈ J' ⟹ a x ∈ carrier R  using True by force
      have prob {ω. ∀x∈J'. hash x ω = a x} =
        real (card {ω ∈ space. ∀x∈J'. eval ω x = a x}) / real (card space)
        by (simp add:M-def measure-pmf-of-set Int-def hash-def)
      also have ... = real (field-size ^ (k − card J')) / real (card space)
        using True by (simp add: poly-cards[OF J'-in-carr card-J'])
      also have
        ... = real field-size ^ (k − card J') / real field-size ^ k
        by (simp add:space-def bounded-degree-polynomials-card)
      also have
        ... = real field-size ^ ((k − 1) ∗ card J') / real field-size ^ (k ∗ card J')
        using card-J' by (simp add:power-add[symmetric] power-mult[symmetric]
            diff-mult-distrib frac-eq-eq add.commute)
      also have
        ... = (real field-size ^ (k − 1)) ^ card J' / (real field-size ^ k) ^ card J'
        by (simp add:power-add power-mult)
      also have
        ... = (∏x∈J'. real (card {ω ∈ space. eval ω x = a x}) / real (card space))
        using a-carr poly-cards-single[OF subsetD[OF J'-in-carr]]
        by (simp add:space-def bounded-degree-polynomials-card power-divide)
      also have ... = (∏x∈J'. prob {ω. hash x ω = a x})
        by (simp add:measure-pmf-of-set M-def Int-def hash-def)
      finally show ?thesis by simp
    next
      case False
      then obtain j where j-def: j ∈ J' a j ∉ carrier R by blast
      have {ω ∈ space. hash j ω = a j} ⊆ {ω ∈ space. hash j ω ∉ carrier R}
        by (rule subsetI, simp add:j-def)
      also have ... ⊆ {} using j-def(1) J'-in-carr hash-range by blast
      finally have b:{ω ∈ space. hash j ω = a j} = {} by simp
      hence real (card ({ω ∈ space. hash j ω = a j})) = 0 by simp
      hence (∏x∈J'. real (card {ω ∈ space. hash x ω = a x})) = 0
        using a(2) prod-zero[OF a(2)] j-def(1) by auto
      moreover have
        {ω ∈ space. ∀x∈J'. hash x ω = a x} ⊆ {ω ∈ space. hash j ω = a j}
        using j-def by blast
      hence {ω ∈ space. ∀x∈J'. hash x ω = a x} = {} using b by blast
      ultimately show ?thesis
        by (simp add:measure-pmf-of-set M-def Int-def prod-dividef)
    qed
  qed
qed

lemma k-wise-indep:
  k-wise-indep-vars k (λ-. discrete) hash (carrier R)
  unfolding k-wise-indep-vars-def using indep by simp

lemma inj-if-degree-1:
  assumes ω ∈ space
```

assumes *degree* ω = *1*
shows *inj-on* (λx. *hash* x ω) (*carrier R*)
using *assms eval-inj-if-degree-1*
by (*simp add*:*M-def space-def bounded-degree-polynomials-def hash-def*)

lemma *uniform*:
assumes $i \in$ *carrier R*
shows *uniform-on* (*hash i*) (*carrier R*)
proof −
have *a*:
$\bigwedge a$. *prob* {ω. *hash i* $\omega \in \{a\}$} = *indicat-real* (*carrier R*) *a* / *real field-size*
by (*subst prob-range*[*OF assms*], *simp add*:*indicator-def*)
show *?thesis*
by (*rule uniform-onI*, *use a M-def* **in** *auto*)
qed

This the main result of this section - the Carter-Wegman hash family is k-universal.

theorem *k-universal*:
k-universal k hash (*carrier R*) (*carrier R*)
using *uniform k-wise-indep* **by** (*simp add*:*k-universal-def*)

end

lemma *poly-hash-familyI*:
assumes *ring R*
assumes *finite* (*carrier R*)
assumes *0* < *k*
shows *poly-hash-family R k*
using *assms*
by (*simp add*:*poly-hash-family-def poly-hash-family-axioms-def*)

lemma *carter-wegman-hash-familyI*:
assumes *field F*
assumes *finite* (*carrier F*)
assumes *0* < *k*
shows *carter-wegman-hash-family F k*
using *assms field.is-ring*[*OF assms*(*1*)] *poly-hash-familyI*
by (*simp add*:*carter-wegman-hash-family-def carter-wegman-hash-family-axioms-def*)

lemma *hash-k-wise-indep*:
assumes *field F* $\wedge$ *finite* (*carrier F*)
assumes *1* $\leq$ *n*
shows
prob-space.k-wise-indep-vars (*pmf-of-set* (*bounded-degree-polynomials F n*)) *n*
(λ-. *pmf-of-set* (*carrier F*)) (*ring.hash F*) (*carrier F*)
proof −
interpret *carter-wegman-hash-family F n*
using *assms carter-wegman-hash-familyI* **by** *force*

have *k-wise-indep-vars n* (λ-. *pmf-of-set* (*carrier F*)) *hash* (*carrier F*)
by (*rule k-wise-indep-vars-compose*[*OF k-wise-indep*], *simp*)
thus *?thesis*
by (*simp add*:*M-def space-def*)
qed

lemma *hash-prob-single*:
assumes *field F* ∧ *finite* (*carrier F*)
assumes $x \in$ *carrier F*
assumes $1 \le n$
assumes $y \in$ *carrier F*
shows
$\mathcal{P}(\omega$ *in pmf-of-set* (*bounded-degree-polynomials F n*). *ring.hash F x* $\omega = y)$
$= 1/($*real* (*card* (*carrier F*)))
proof −
interpret *carter-wegman-hash-family F n*
using *assms carter-wegman-hash-familyI* **by** *force*
show *?thesis*
using *prob-single*[*OF assms*(*2*,*4*)] **by** (*simp add*:*M-def space-def*)
qed

end

4 Finite Fields

theory *Field*
imports
Finite-Fields.Ring-Characteristic
HOL−Algebra.Ring-Divisibility
HOL−Algebra.IntRing
begin

In some applications it is more convenient to work with natural numbers instead of *ZFact p* whose elements are cosets. To support that use case the following definition introduces an additive and multiplicative structure on {..<*p*}. After verifying that the function *zfact-iso* and its inverse are homomorphisms, the ring and field property can be transfered from *ZFact p* to to the structure on {..<*p*}.

lemma *zfact-iso-0*:
assumes $n > 0$
shows *zfact-iso n 0* = $\mathbf{0}_{ZFact\ (int\ n)}$
proof −
let *?I* = $Idl_{\mathcal{Z}}$ {*int n*}
have *ideal-I*: *ideal ?I* $\mathcal{Z}$
by (*simp add*: *int.genideal-ideal*)

interpret *i*:*ideal ?I* $\mathcal{Z}$ **using** *ideal-I* **by** *simp*
interpret *s*:*ring-hom-ring* $\mathcal{Z}$ *ZFact* (*int n*) $(+>_{\mathcal{Z}})$ *?I*

using *i.rcos-ring-hom-ring ZFact-def* **by** *auto*

show *?thesis*
by (*simp add*:*zfact-iso-def ZFact-def*)
qed

lemma *zfact-prime-is-field*:
assumes *Factorial-Ring.prime* (*p* :: *nat*)
shows *field* (*ZFact* (*int p*))
using *zfact-prime-is-finite-field*[*OF assms*] *finite-field-def* **by** *auto*

definition *mod-ring* :: *nat* => *nat ring*
where *mod-ring n* = (|
carrier = {..<*n*},
mult = (λ *x y*. (*x* * *y*) *mod n*),
one = *1*,
zero = *0*,
add = (λ *x y*. (*x* + *y*) *mod n*) |)

definition *zfact-iso-inv* :: *nat* ⇒ *int set* ⇒ *nat* **where**
zfact-iso-inv p = *inv-into* {..<*p*} (*zfact-iso p*)

lemma *zfact-iso-inv-0*:
assumes *n-ge-0*: *n* > *0*
shows *zfact-iso-inv n* $\mathbf{0}_{ZFact\ (int\ n)}$ = *0*
unfolding *zfact-iso-inv-def zfact-iso-0*[*OF n-ge-0*, *symmetric*] **using** *n-ge-0*
by (*rule inv-into-f-f*[*OF zfact-iso-inj*], *simp add*:*mod-ring-def*)

lemma *zfact-coset*:
assumes *n-ge-0*: *n* > *0*
assumes *x* ∈ *carrier* (*ZFact* (*int n*))
defines *I* ≡ $Idl_{\mathcal{Z}}$ {*int n*}
shows *x* = *I* $+>_{\mathcal{Z}}$ (*int* (*zfact-iso-inv n x*))
proof −
have *x* ∈ *zfact-iso n* ‘ {..<*n*}
using *assms zfact-iso-ran* **by** *simp*
hence *zfact-iso n* (*zfact-iso-inv n x*) = *x*
unfolding *zfact-iso-inv-def* **by** (*rule f-inv-into-f*)
thus *?thesis* **unfolding** *zfact-iso-def I-def* **by** *blast*
qed

lemma *zfact-iso-inv-is-ring-iso*:
assumes *n-ge-1*: *n* > *1*
shows *zfact-iso-inv n* ∈ *ring-iso* (*ZFact* (*int n*)) (*mod-ring n*)
proof (*rule ring-iso-memI*)
interpret *r*:*cring* (*ZFact* (*int n*))
using *ZFact-is-cring* **by** *simp*

define *I* **where** *I* = $Idl_{\mathcal{Z}}$ {*int n*}

have *n-ge-0*: $n > 0$ **using** *n-ge-1* **by** *simp*

interpret *i*:*ideal I* $\mathcal{Z}$
  unfolding *I-def* **using** *int.genideal-ideal* **by** *simp*

interpret *s*:*ring-hom-ring* $\mathcal{Z}$ *ZFact* (*int n*) ($+>_{\mathcal{Z}}$) *I*
 using *i.rcos-ring-hom-ring ZFact-def I-def* **by** *auto*

show
  $\bigwedge x.\ x \in carrier\ (ZFact\ (int\ n)) \Longrightarrow zfact\text{-}iso\text{-}inv\ n\ x \in carrier\ (mod\text{-}ring\ n)$
proof −
  fix *x*
  assume $x \in carrier\ (ZFact\ (int\ n))$
  hence $zfact\text{-}iso\text{-}inv\ n\ x \in \{..<n\}$
    unfolding *zfact-iso-inv-def*
    using *zfact-iso-ran*[*OF n-ge-0*] *inv-into-into* **by** *metis*

  thus $zfact\text{-}iso\text{-}inv\ n\ x \in carrier\ (mod\text{-}ring\ n)$
    unfolding *mod-ring-def* **by** *simp*
qed

show $\bigwedge x\ y.\ x \in carrier\ (ZFact\ (int\ n)) \Longrightarrow y \in carrier\ (ZFact\ (int\ n)) \Longrightarrow$
  $zfact\text{-}iso\text{-}inv\ n\ (x \otimes_{ZFact\ (int\ n)} y) =$
  $zfact\text{-}iso\text{-}inv\ n\ x \otimes_{mod\text{-}ring\ n} zfact\text{-}iso\text{-}inv\ n\ y$
proof −
  fix *x y*
  assume *x-carr*: $x \in carrier\ (ZFact\ (int\ n))$
  define x' **where** $x' = zfact\text{-}iso\text{-}inv\ n\ x$
  assume *y-carr*: $y \in carrier\ (ZFact\ (int\ n))$
  define y' **where** $y' = zfact\text{-}iso\text{-}inv\ n\ y$
  have $x \otimes_{ZFact\ (int\ n)} y = (I +>_{\mathcal{Z}} (int\ x')) \otimes_{ZFact\ (int\ n)} (I +>_{\mathcal{Z}} (int\ y'))$
    unfolding *x′-def y′-def*
    using *x-carr y-carr zfact-coset*[*OF n-ge-0*] *I-def* **by** *simp*
  also have ... $= (I +>_{\mathcal{Z}} (int\ x' * int\ y'))$
    by *simp*
  also have ... $= (I +>_{\mathcal{Z}} (int\ ((x' * y')\ mod\ n)))$
    unfolding *I-def zmod-int* **by** (*rule int-cosetI*[*OF n-ge-0*],*simp*)
  also have ... $= (I +>_{\mathcal{Z}} (x' \otimes_{mod\text{-}ring\ n} y'))$
    unfolding *mod-ring-def* **by** *simp*
  also have ... $= zfact\text{-}iso\ n\ (x' \otimes_{mod\text{-}ring\ n} y')$
    unfolding *zfact-iso-def I-def* **by** *simp*
  finally have *a*:$x \otimes_{ZFact\ (int\ n)} y = zfact\text{-}iso\ n\ (x' \otimes_{mod\text{-}ring\ n} y')$
    by *simp*
  have *b*:$x' \otimes_{mod\text{-}ring\ n} y' \in \{..<n\}$
    using *mod-ring-def n-ge-0* **by** *auto*
  have $zfact\text{-}iso\text{-}inv\ n\ (zfact\text{-}iso\ n\ (x' \otimes_{mod\text{-}ring\ n} y')) = x' \otimes_{mod\text{-}ring\ n} y'$
    unfolding *zfact-iso-inv-def*
    by (*rule inv-into-f-f*[*OF zfact-iso-inj*[*OF n-ge-0*] *b*])

thus
zfact-iso-inv n $(x \otimes_{ZFact\ (int\ n)}\ y) =$
zfact-iso-inv n x $\otimes_{mod\text{-}ring\ n}$ *zfact-iso-inv n y*
using *a x′-def y′-def* **by** *simp*
qed

show $\bigwedge x\ y.\ x \in$ *carrier* (*ZFact* (*int n*)) $\Longrightarrow y \in$ *carrier* (*ZFact* (*int n*)) $\Longrightarrow$
zfact-iso-inv n $(x \oplus_{ZFact\ (int\ n)}\ y) =$
zfact-iso-inv n x $\oplus_{mod\text{-}ring\ n}$ *zfact-iso-inv n y*
proof −
fix *x y*
assume *x-carr*: $x \in$ *carrier* (*ZFact* (*int n*))
define x' **where** $x' =$ *zfact-iso-inv n x*
assume *y-carr*: $y \in$ *carrier* (*ZFact* (*int n*))
define y' **where** $y' =$ *zfact-iso-inv n y*
have $x \oplus_{ZFact\ (int\ n)}\ y = (I +>_{\mathcal{Z}} (int\ x')) \oplus_{ZFact\ (int\ n)} (I +>_{\mathcal{Z}} (int\ y'))$
unfolding *x′-def y′-def*
using *x-carr y-carr zfact-coset*[*OF n-ge-0*] *I-def* **by** *simp*
also have ... $= (I +>_{\mathcal{Z}} (int\ x' + int\ y'))$
by *simp*
also have ... $= (I +>_{\mathcal{Z}} (int\ ((x' + y')\ mod\ n)))$
unfolding *I-def zmod-int* **by** (*rule int-cosetI*[*OF n-ge-0*],*simp*)
also have ... $= (I +>_{\mathcal{Z}} (x' \oplus_{mod\text{-}ring\ n}\ y'))$
unfolding *mod-ring-def* **by** *simp*
also have ... $=$ *zfact-iso n* $(x' \oplus_{mod\text{-}ring\ n}\ y')$
unfolding *zfact-iso-def I-def* **by** *simp*
finally have *a*:$x \oplus_{ZFact\ (int\ n)}\ y =$ *zfact-iso n* $(x' \oplus_{mod\text{-}ring\ n}\ y')$
by *simp*
have *b*:$x' \oplus_{mod\text{-}ring\ n}\ y' \in \{..<n\}$
using *mod-ring-def n-ge-0* **by** *auto*
have *zfact-iso-inv n* (*zfact-iso n* $(x' \oplus_{mod\text{-}ring\ n}\ y')) = x' \oplus_{mod\text{-}ring\ n}\ y'$
unfolding *zfact-iso-inv-def*
by (*rule inv-into-f-f*[*OF zfact-iso-inj*[*OF n-ge-0*] *b*])
thus
zfact-iso-inv n $(x \oplus_{ZFact\ (int\ n)}\ y) =$
zfact-iso-inv n x $\oplus_{mod\text{-}ring\ n}$ *zfact-iso-inv n y*
using *a x′-def y′-def* **by** *simp*
qed

have $\mathbf{1}_{ZFact\ (int\ n)} =$ *zfact-iso n* $(\mathbf{1}_{mod\text{-}ring\ n})$
by (*simp add*:*zfact-iso-def ZFact-def I-def*[*symmetric*] *mod-ring-def*)

thus *zfact-iso-inv n* $\mathbf{1}_{ZFact\ (int\ n)} = \mathbf{1}_{mod\text{-}ring\ n}$
unfolding *zfact-iso-inv-def mod-ring-def*
using *inv-into-f-f*[*OF zfact-iso-inj*] *n-ge-1* **by** *simp*

show *bij-betw* (*zfact-iso-inv n*) (*carrier* (*ZFact* (*int n*))) (*carrier* (*mod-ring n*))
using *zfact-iso-inv-def mod-ring-def zfact-iso-bij*[*OF n-ge-0*] *bij-betw-inv-into*

by *force*
qed

lemma *mod-ring-finite*:
finite (*carrier* (*mod-ring* *n*))
by (*simp* *add*:*mod-ring-def*)

lemma *mod-ring-carr*:
$x \in$ *carrier* (*mod-ring* *n*) $\longleftrightarrow$ $x < n$
by (*simp* *add*:*mod-ring-def*)

lemma *mod-ring-is-cring*:
assumes *n-ge-1*: $n > 1$
shows *cring* (*mod-ring* *n*)
proof −
have *n-ge-0*: $n > 0$ **using** *n-ge-1* **by** *simp*

interpret *cring* *ZFact* (*int* *n*)
using *ZFact-is-cring* **by** *simp*

have *cring* ((*mod-ring* *n*) $(\!|$ *zero* := *zfact-iso-inv* *n* $\mathbf{0}_{ZFact\ (int\ n)}$ $|\!)$)
by (*rule* *ring-iso-imp-img-cring*[*OF* *zfact-iso-inv-is-ring-iso*[*OF* *n-ge-1*]])
moreover have
(*mod-ring* *n*) $(\!|$ *zero* := *zfact-iso-inv* *n* $\mathbf{0}_{ZFact\ (int\ n)}$ $|\!)$ = *mod-ring* *n*
using *zfact-iso-inv-0*[*OF* *n-ge-0*]
by (*simp* *add*:*mod-ring-def*)
ultimately show *?thesis* **by** *simp*
qed

lemma *zfact-iso-is-ring-iso*:
assumes *n-ge-1*: $n > 1$
shows *zfact-iso* *n* $\in$ *ring-iso* (*mod-ring* *n*) (*ZFact* (*int* *n*))
proof −
have *r*:*ring* (*ZFact* (*int* *n*))
using *ZFact-is-cring* *cring.axioms*(*1*) **by** *blast*

interpret *s*: *ring* (*mod-ring* *n*)
using *mod-ring-is-cring* *cring.axioms*(*1*) *n-ge-1* **by** *blast*
have *n-ge-0*: $n > 0$ **using** *n-ge-1* **by** *linarith*

have
inv-into (*carrier* (*ZFact* (*int* *n*))) (*zfact-iso-inv* *n*)
$\in$ *ring-iso* (*mod-ring* *n*) (*ZFact* (*int* *n*))
using *ring-iso-set-sym*[*OF* *r* *zfact-iso-inv-is-ring-iso*[*OF* *n-ge-1*]] **by** *simp*
moreover have $\bigwedge x.\ x \in$ *carrier* (*mod-ring* *n*) $\Longrightarrow$
inv-into (*carrier* (*ZFact* (*int* *n*))) (*zfact-iso-inv* *n*) *x* = *zfact-iso* *n* *x*
proof −
fix *x*
assume $x \in$ *carrier* (*mod-ring* *n*)

hence $x \in \{..<n\}$ **by** (*simp add*:*mod-ring-def*)
thus *inv-into* (*carrier* (*ZFact* (*int n*))) (*zfact-iso-inv n*) x = *zfact-iso n x*
unfolding *zfact-iso-inv-def*
by (*simp add*:*inv-into-inv-into-eq*[*OF zfact-iso-bij*[*OF n-ge-0*]])
qed

ultimately show *?thesis*
using *s.ring-iso-restrict* **by** *blast*
qed

If p is a prime than *mod-ring* p is a field:

lemma *mod-ring-is-field*:
assumes*Factorial-Ring.prime p*
shows *field* (*mod-ring p*)
proof −
have *p-ge-0*: $p > 0$ **using** *assms prime-gt-0-nat* **by** *blast*
have *p-ge-1*: $p > 1$ **using** *assms prime-gt-1-nat* **by** *blast*

interpret *field ZFact* (*int p*)
using *zfact-prime-is-field*[*OF assms*] **by** *simp*

have *field* ((*mod-ring p*) ⦇ *zero* := *zfact-iso-inv p* $\mathbf{0}_{ZFact\ (int\ p)}$ ⦈)
by (*rule ring-iso-imp-img-field*[*OF zfact-iso-inv-is-ring-iso*[*OF p-ge-1*]])

moreover have
(*mod-ring p*) ⦇ *zero* := *zfact-iso-inv p* $\mathbf{0}_{ZFact\ (int\ p)}$ ⦈ = *mod-ring p*
using *zfact-iso-inv-0*[*OF p-ge-0*]
by (*simp add*:*mod-ring-def*)
ultimately show *?thesis* **by** *simp*
qed

end

References

[1] E. Karayel. Interpolation polynomials (in hol-algebra). *Archive of Formal Proofs*, Jan. 2022. https://isa-afp.org/entries/Interpolation_Polynomials_HOL_Algebra.html, Formal proof development.

[2] M. Thorup and Y. Zhang. Tabulation based 5-universal hashing and linear probing. In *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, ALENEX '10, pages 62–76, USA, 2010. Society for Industrial and Applied Mathematics.

[3] S. P. Vadhan. Pseudorandomness. *Foundations and Trends®in Theoretical Computer Science*, 7(1-3):1–336, 2012.

[4] M. N. Wegman and J. L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981.